Programación web

CSS AVANZADO - LAYOUT

Flexbox

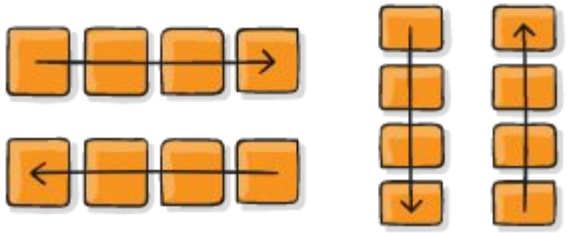
- Es un método para ayudarnos a distribuir el espacio entre los ítems de nuestra interfaz, y personalizar la alineación entre ellos

- El sistema flex consta de dos grandes componentes, a los cuales les podemos definir diferentes atributos:
  - **El contenedor**
  - **Los ítems adentro del contenedor**

Flexbox

- Flexbox maneja el layout en una sola dimensión a la vez — ya sea como fila o como columna

- Cuando trabajamos con flexbox necesitamos pensar en términos de dos ejes:
  - el **eje principal**, que está definido por la propiedad **flex-direction**
    - row
    - row-reverse
    - column
    - column-reverse



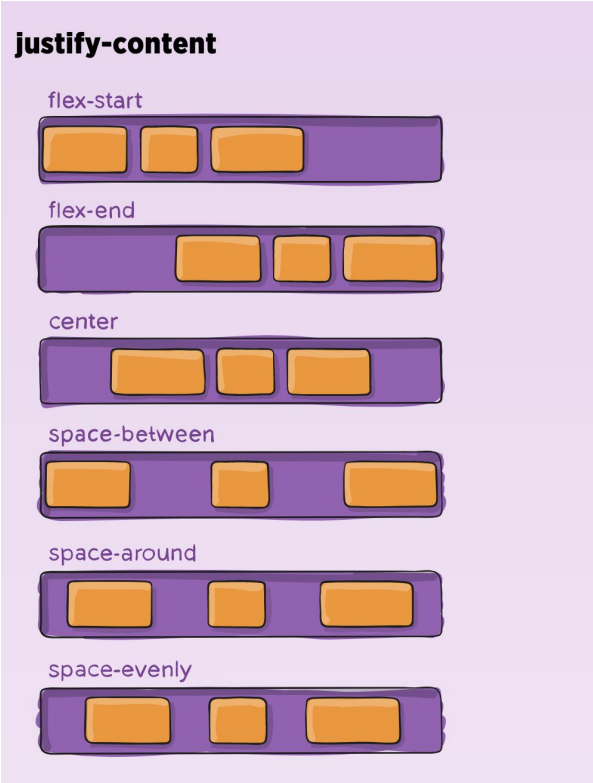
  - el **eje cruzado** va perpendicular al eje principal, y por lo tanto si flex-direction es row o row-reverse el eje cruzado irá por las columnas, y viceversa

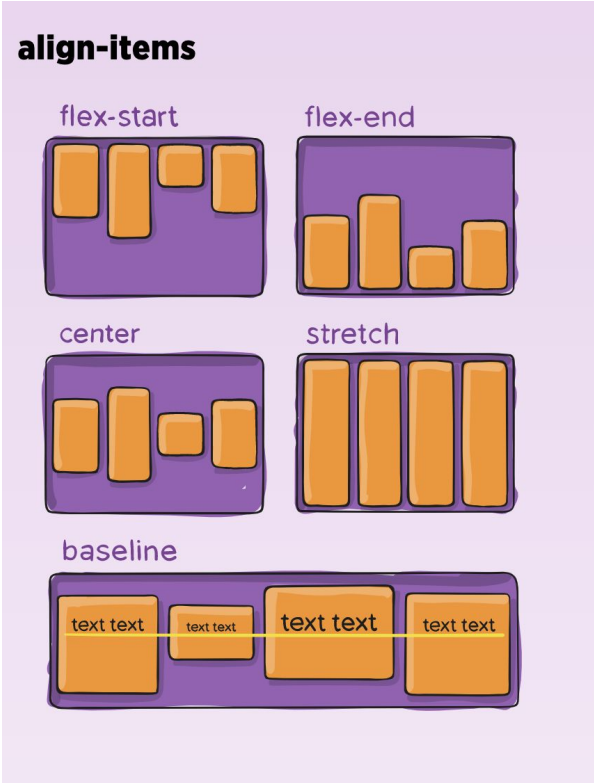
Container

```
.flex-container {
    display: flex;
}
```

- **flex-direction:** row | row-reverse | column | column-reverse;
- **flex-wrap:** nowrap | wrap | wrap-reverse;
- **flex-flow:** column wrap;
- **justify-content:** flex-start | flex-end | center | space-between | space-around | space-evenly | start | end | left | right ... + safe | unsafe;
- **align-items:** stretch | flex-start | flex-end | center | baseline | first baseline | last baseline | start | end | self-start | self-end + ... safe | unsafe;
- **align-content:** flex-start | flex-end | center | space-between | space-around | space-evenly | stretch | start | end | baseline | first baseline | last baseline + ... safe | unsafe;

Container


justify-content
flex-start
flex-end
center
space-between
space-around
space-evenly


align-items
flex-start
flex-end
center
stretch
baseline
text text
text text
text text
text text

Items

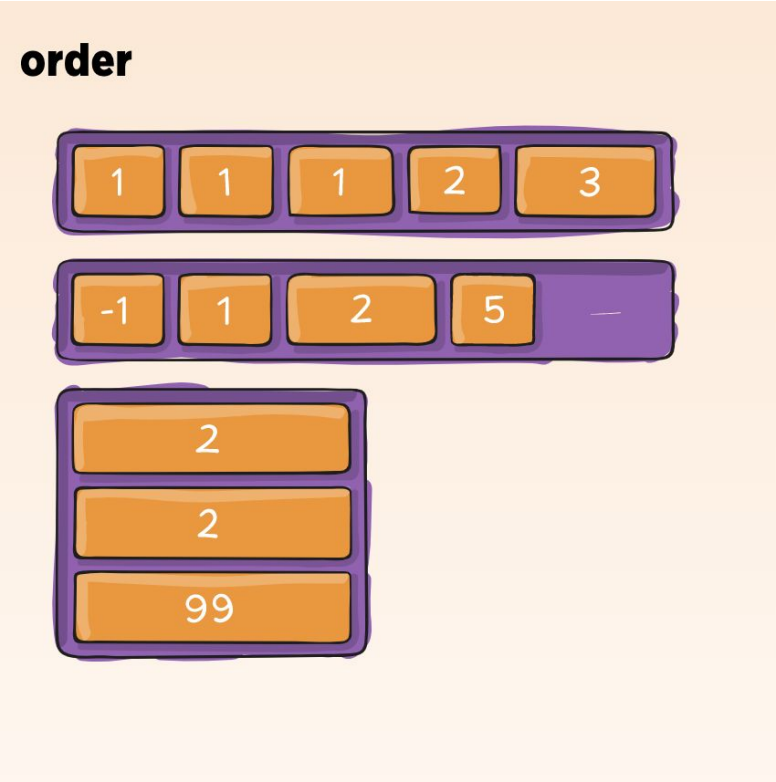
```
<div class="flex-container">
    <div class="flex-item">One</div>
    <div class="flex-item">Two</div>
    <div class="flex-item">Three</div>
</div>
```

```
.flex-item {

}
```

- order
- flex-grow
- flex-shrink
- flex-basis
- flex
- align-self

Items



Apliquemos Flex a nuestro proyecto

Agreguemos una nueva página!

Media queries

- Nos permiten definir diferentes reglas de estilo para diferentes tipos de medios.
  - Originalmente se usaba para distinguir pantallas de computadora, impresoras, dispositivos portátiles, dispositivos tipo televisión, etc.
  - Ahora, en lugar de buscar un tipo de dispositivo, miramos la capacidad del dispositivo.

- Son una herramienta para ayudarnos a que nuestra página sea **responsive**, es decir, que su contenido se adapte al dispositivo.

- Podemos verificar varias cosas:
  - ancho y alto de la ventana gráfica
  - ancho y alto del dispositivo
  - orientación (la tablet / teléfono está en modo horizontal o vertical?)
  - resolución

Media queries

```
@media screen (min-width: 320px) and (max-width: 768px)
```

TIPO DE DISPOSITIVO | CONDICIÓN | OPERADOR | CONDICIÓN

```
/* Cuando el navegador tiene al menor 600px o más */
@media (min-width: 600px) {
    .element {
      /* estilos */
    }
}
```

Media queries

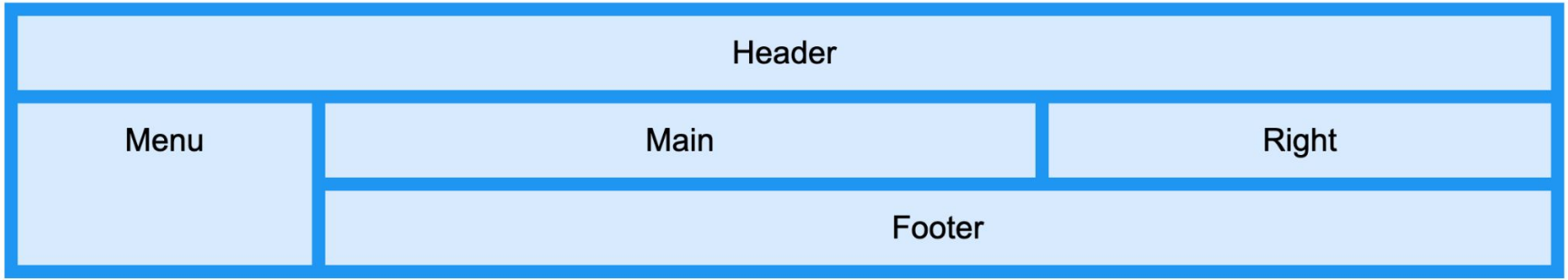
- Tipos de dispositivos:
  - all: Todos los dispositivos
  - print: documentos que se ven en una vista previa de impresión o cualquier medio que divida el contenido en páginas destinadas a imprimir
  - screen: dispositivos que tengan una pantalla
  - speech: dispositivos que leen el contenido, como un lector de pantalla

- Algunos ejemplos de expresiones:
  - min-width: 1024px
  - max-width: 400px
  - min-height: 800px
  - max-height: 400px
  - orientation: portrait
  - orientation: landscape

Apliquemos media queries a la nueva página

Grid

- Es otra forma de acomodar los elementos de nuestra página, como alternativa a Flexbox
- Nos permite organizar los elementos en columnas y filas
- La diferencia básica entre Grid y Flexbox es que Flexbox se creó para diseños de una dimensión, en una fila o una columna. En cambio Grid, se pensó para el diseño bidimensional, en varias filas y columnas al mismo tiempo



Grid

- También tenemos un contenedor con sus elementos

```
<div class="grid-container">
    <div class="grid-item">1</div>
    <div class="grid-item">2</div>
    <div class="grid-item">3</div>
</div>
```

- Lo convertimos en un contenedor Grid de esta forma:

```
.grid-container {
    display: grid
}
```

Layout 1

Al contenedor le ponemos **grid-template-columns**

- grid-template-columns: 40px 50px 100px 50px 40px;
- grid-template-columns: 40px 50px auto 50px 40px;
- grid-template-columns: 25% 25% 25% 25%;
- grid-template-columns: repeat(4, 25%);
- grid-template-columns: 100px 1fr 1fr 1fr;
- grid-template-columns: 100px repeat(3, 1fr);

* repeat(cantidad de filas/columnas, la longitud que queremos);
* fr = fracción del espacio libre que hay

```
.container {
  grid-template-columns: 40px 50px auto 50px 40px;
  grid-template-rows: 25% 100px auto;
}
```

A los items les ponemos **grid-column / grid-row**:

```
.item {
  grid-column: <start-line> / <end-line>;
  grid-row: <start-line> / <end-line>;
}
```

- start-line/end-line: número de línea

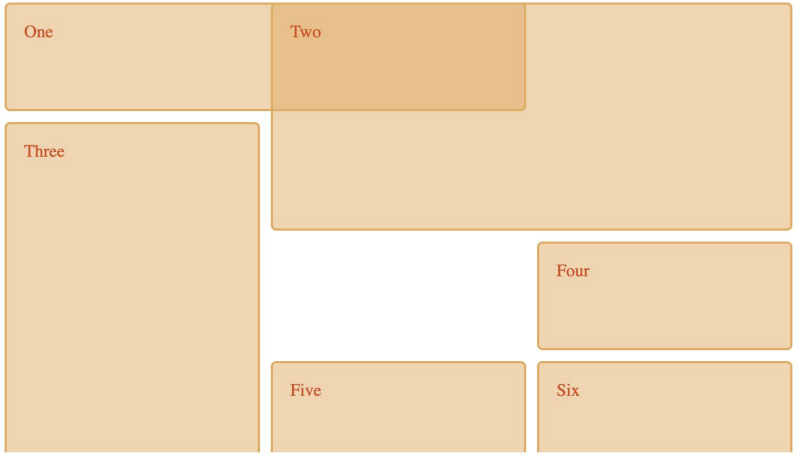
```
.item {
  grid-column: <start-line> / span <value>;
  grid-row: <start-line> / span <value>;
}
```

- value: cuántas celdas tomar

```html
<div class="wrapper">
    <div class="one">One</div>
    <div class="two">Two</div>
    <div class="three">Three</div>
    <div class="four">Four</div>
    <div class="five">Five</div>
    <div class="six">Six</div>
</div>
```

```css
.wrapper {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    grid-gap: 10px;
}
```

```css
.one {
    grid-column: 1 / 3;
    grid-row: 1;
}
.two {
    grid-column: 2 / 4;
    grid-row: 1 / 3;
}
.three {
    grid-column: 1;
    grid-row: 2 / 5;
}
.four {
    grid-column: 3;
    grid-row: 3;
}
.five {
    grid-column: 2;
    grid-row: 4;
}
.six {
    grid-column: 3;
    grid-row: 4;
}
```



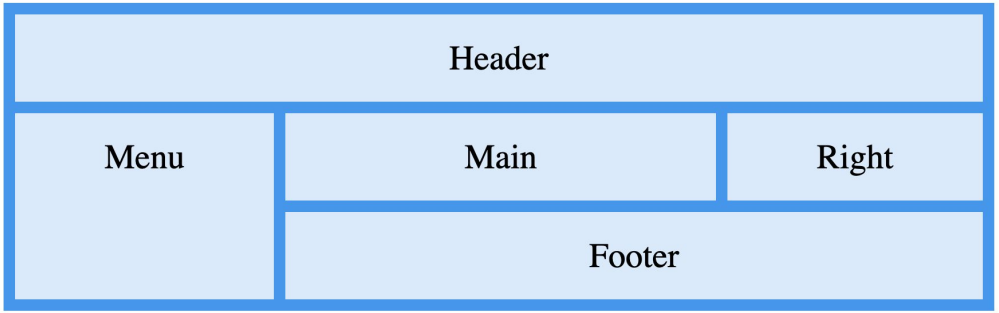
Layout 2

A cada ítem le asignamos un nombre de área con la propiedad **grid-area**:

```
.item1 { grid-area: header; }
.item2 { grid-area: menu; }
.item3 { grid-area: main; }
.item4 { grid-area: right; }
.item5 { grid-area: footer; }
```

En el contenedor definimos cómo queremos organizar las áreas con la propiedad **grid-template-areas**:

```
.grid-container {
    display: grid;
    grid-template-areas:
      'header header header header'
      'menu main main right'
      'menu footer footer footer';
    grid-gap: 10px;
    background-color: #2196F3;
  }
```



Algunas propiedades

- **Contenedor:**
  - column-gap
  - row-gap
  - **grid-gap**
  - **justify-items**: start | end | center | stretch;
    - alinea los elementos a nivel filas
    - stretch es el default
  - **align-items**: start | end | center | stretch;
    - alinea los elementos a nivel columnas
    - stretch es el default
  - etc.

- **Items:**
  - justify-self: start | end | center | stretch;
  - align-self: start | end | center | stretch;
  - etc.

Apliquemos Grid a nuestro proyecto

Dupliquemos nuestro CSS y hagamos uno nuevo con este tipo de layout.

Después lo podremos intercambiar desde el HTML.

Responsive

- Grid por su forma de organizar los elementos, está más orientado hacia ser responsive que Flexbox.
- Por ejemplo, si ponemos el width de nuestras imágenes en un % de la celda que ocupa, en vez de píxeles, vamos a ver como esta se adapta según el tamaño de la pantalla.
- **minmax(min, max)**
  - El tamaño del componente puede variar pero solo entre min y max
  - Si max < min, entonces max es ignorado y se trata a minmax(min,max) como min
  - minmax(100px, 200px)
  - minmax(200px, 50%)
  - minmax(200px, 1fr)

Responsive

```
grid-template-columns: repeat( 12, minmax(250px, 1fr) );
```

- En este caso, las 12 columnas van a tener un valor entre 250px y 1fr.
- Pero siempre van a ser 12 columnas.

- Si queremos que la cantidad de columnas también se adapte dinámicamente, podemos usar:

```
grid-template-columns: repeat( auto-fit, minmax(250px, 1fr) );
```

- **auto-fit** adapta las columnas disponibles actualmente en el espacio, expandiéndolas para que ocupen cualquier espacio disponible.

- **auto-fill** llena la fila con tantas columnas como puedan caber, por lo tanto, crea columnas vacías cada vez que cabe una nueva columna.

Responsive

Hagamos nuestro proyecto Grid responsive